Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Treći, septembar 2015.
Datum: 25.8.2015.

Treći kolokvijum iz Operativnih sistema 2

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 1,5 sat. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____%

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Ocenjivanje unutar potpitanja je po sistemu "sve ili ništa", odnosno nema parcijalnih poena. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**.

1. (10 poena) RAID strukture

Neki sistem implementira *block-striping* RAID10 funkcionalnost u softveru.

Svi diskovi su identični, imaju isti broj blokova (taj broj blokova na jednom disku vraća funkcija `getNumOfBlocks()`). Postoji *n* parova diskova koji se ogledaju, pri čemu broj *n* vraća funkcija `getNumOfDisks()`.

Definisan je globalni niz `disks` od *n* elemenata, gde svaki element niza predstavlja jedan par diskova objektom sledeće klase:

```
class DiskPair {
public:
  int isOK (int diskNo);
  int getLastRead ();
  void putReadRequest (int diskNo, unsigned blockNo, void* buffer);
};
```

U operacijama ove klase celobrojni argument `diskNo` može imati vrednost 0 ili 1 koja ukazuje na jedan ili drugi disk u paru. Funkcija `isOK()` vraća 1 ako je dati disk u paru ispravan i funkcionalan, a 0 ako je otkazao ili nije instaliran. Funkcija `getLastRead()` vraća broj (0 ili 1) diska u paru kome je poslednjem bio izdat zahtev za čitanje. Funkcija `putReadRequest`() zadaje novi zahtev za čitanje na disku `diskNo` u paru sa datim parametrima – redni broj bloka na tom fizičkom disku i adresa bafera u memoriji za čitanje, i odmah vraća kontrolu pozivaocu.

Implementirati funkciju:

```
int readBlock (unsigned blk, void* buffer);
```

koja treba da zada novi zahtev za čitanje sa logičkog bloka broj `blk` (logički broj za celu RAID10 strukturu kao jedinstven virtuelni disk) u bafer čija je adresa data. Ako su oba diska u paru na koji se preslikava dati logički blok ispravna, zahteve za čitanjem treba zadavati diskovima u paru naizmenično, radi raspodele opterećenja i paralelizacije operacija. U slučaju uspeha, operacija treba da vrati 0. Ukoliko je redni broj bloka prekoračio dozvoljen opseg, operacija treba da vrati -1. Ukoliko su oba diska u paru na koji se preslikava dati blok neispravna, operacija treba da vrati -2.

Rešenje:

2. (10 poena) Operativni sistem Linux

Napisati *shell script* koji treba da u direktorijumu zadatom kao prvi parametar pronađe sve fajlove sa ekstenzijom .cpp (direktorijum se pretražuje do proizvoljne dubine) i u svakom pronađenom fajlu ukloni sve komentare u jednoj liniji. Komentar počinje sa // i završava se na kraju reda. Smatrati da imena fajlova ne mogu da sadrže razmake. Ukoliko prosleđeni prvi parametar nije direktorijum ili ukoliko broj parametara nije odgovarajući, treba ispisati poruku o grešci.

Rešenje:

3. (10 poena) Operativni sistem Linux

Na jeziku C/C++, koristeći mehanizam deljene memorije i semafora kod operativnog sistema Linux, napisati program koji pravi procese koji međusobno komuniciraju preko razmene poruka (nije dozvoljeno koristiti mehanizam prosleđivanja poruka putem poštanskog sandučeta kod operativnog sistema Linux). Broj procesa koje je potrebno pokrenuti je dat kao konstanta `N`. Vrednost ključa za dohvatanje odgovarajućeg segmenata deljene memorije je data kao konstanta `keyMem`. Vrednost ključa za dohvatanje odgovarajućeg semafora je data kao konstanta `keySem`. Svaki kreirani proces ima jedinstveni identifikator procesa:

```
typedef unsigned Id;
```

Programi treba da komuniciraju putem poruka koji su tipa:

```
struct Msg {
  Id receiver;
  ...
};
```

gde je polje `receiver` identifikator procesa koji treba da primi poruku. Broj poruka koji svaki kreirani proces treba da napravi i pošalje je dat kao konstanta `NUM`.

```
void getMsg(struct Msg *msg, Id id);
```

gde parametar `msg` pokazuje na memorijsku lokaciju gde poruku treba smestiti, dok je parameter `id` identifikator procesa koji pravi poruku.

Procesi treba da šalju poruke preko bafera u deljenoj memoriji u koji može da se smesti `SIZE` poruka. Svaki proces prvo proverava da li ima mesta u baferu i ako ima, stavlja jednu poruku. Zatim proces treba da proveri da li se na početku bafera nalaze poruke za njega i ako se nalaze, da ih pročita. Taj posao svaki proces treba da ponavlja dok ne pošalje sve poruke. Proces treba da završi svoje izvršavanje kad primi sve poruke i kad oslobodi sve sistemske resurse. Nije potrebno proveravati uspešnost sistemskih poziva za deljenu memoriju i semafore. Ako je neki deo koda indentičan kao kod dat na vežbama, nije ga potrebno pisati, već samo navesti odakle je preuzet.

Rešenje: